

# Pros and cons of various layer management tools

Yocto Summit 2020

Tomasz Żyjewski



- Introduction
- Yocto layered model
- Presentation goal
- Description of example project
- Layer management - git submodules
- Layer management - repo
- Layer management - combo-layer
- Layer management - kas
- Summary and conclusions



✉ tomasz.zyjewski@3mdeb.com

- Embedded Systems Engineer
- over 1 year Embedded Linux / Yocto experience
- aspiring Yocto contributor



3mdeb is a firmware and embedded systems development company founded by Piotr Król and headquartered in Gdańsk, Poland. We combine the collaborative creativity of the open-source community with the reliable strength of an enterprise-grade solutions provider.

- Starting a new project can be like
  - prepare description of requirements
  - transforming it into list of tools and packages
  - in case of Yocto, every package is delivered by corresponding recipe

At this moment we can decide to provide all required recipes from our main repository or to search and reuse existing recipes from existing layers.

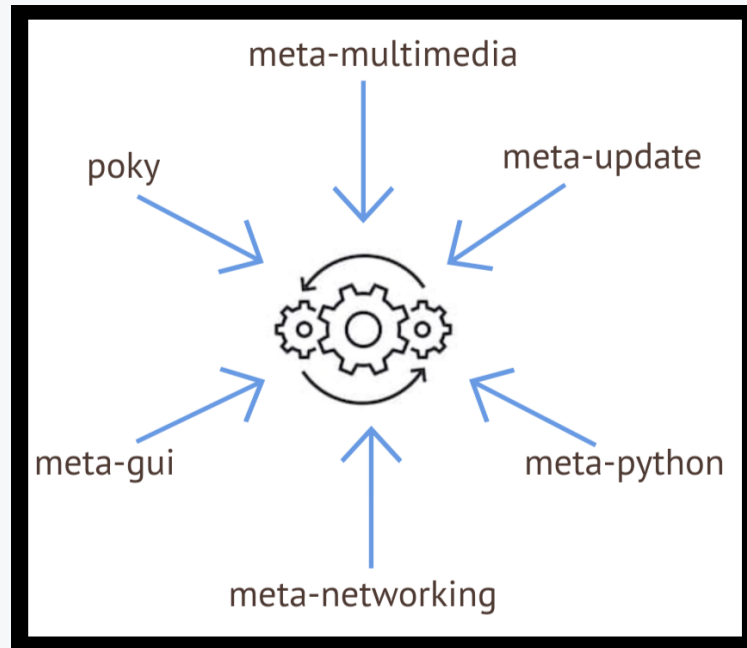
<a href="#">meta-aarch64</a>	AArch64 (64-bit ARM) architecture support	Machine (BSP)	<a href="https://git.linaro.org/openembedded/meta-linaro.git">git://git.linaro.org/openembedded/meta-linaro.git</a>
<a href="#">meta-acme</a>	Acme Systems Yocto meta layer	Machine (BSP)	<a href="https://github.com/myfreescalewebpage/meta-acme">https://github.com/myfreescalewebpage/meta-acme</a>
<a href="#">meta-allwinner-hx</a>	Meta layer for all allwinner H2/H3/H5 boards	Machine (BSP)	<a href="https://gitlab.com/dimtas/meta-allwinner-hx">https://gitlab.com/dimtas/meta-allwinner-hx</a>
<a href="#">meta-altera</a>	Altera SoC BSP layer	Machine (BSP)	<a href="https://github.com/kraj/meta-altera">https://github.com/kraj/meta-altera</a>
<a href="#">meta-amarula-engicam</a>	Yocto Meta BSP layer for Engicam boards	Machine (BSP)	<a href="https://github.com/amarula/meta-amarula-engicam.git">https://github.com/amarula/meta-amarula-engicam.git</a>
<a href="#">meta-amd</a>	AMD board support common layer (official)	Machine (BSP)	<a href="https://git.yoctoproject.org/meta-amd">git://git.yoctoproject.org/meta-amd</a>
<a href="#">meta-amd-falconx86</a>	AMD Falcon board support layer (official)	Machine (BSP)	<a href="https://git.yoctoproject.org/meta-amd">git://git.yoctoproject.org/meta-amd</a>
<a href="#">meta-arduino</a>	Board Support for the Arduino Yún	Machine (BSP)	<a href="https://gitlab.com/toganlabs/meta-arduino">https://gitlab.com/toganlabs/meta-arduino</a>

- So what are the layers in the Yocto Project?
  - one of the many features
  - group related functionalities into separate bundles
  - can be added to any project at any time (but some layers have dependencies)
  - reduces the complexity and redundancy of a project

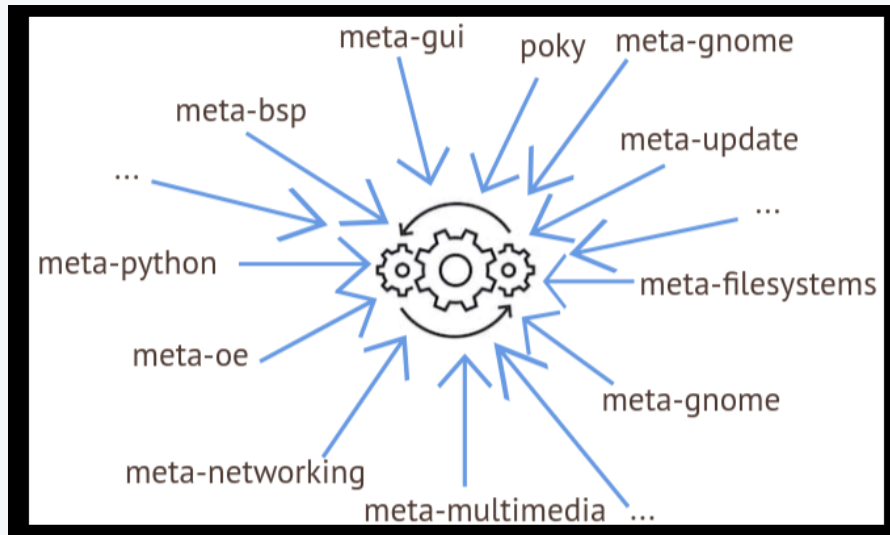
From the files perspective, layers are some repositories which contains meta-data like recipes or configuration files.

Layers can and should be used to logically break down information about our build.

- A good prepared layer is a layer that contains the smallest set of metadata
- This also means that the smallest project can consists of several layers



- The more complex project, the more layers we need to use



- What should we do now? How to start?

Fortunately, the Yocto layered model also allows to create user friendly tools for managing layers. In this presentation I would like to present four such tools

- Use of `git submodules`
- Use of the Google `repo` tool
- `combo-layer` script from poky
- `kas` tool from Siemens repository

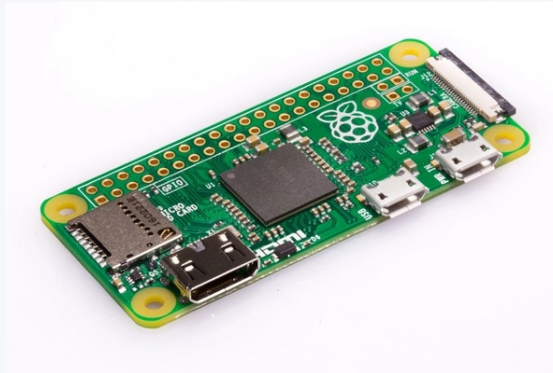
The main goals of this presentation

- A list of advantages and disadvantages of given solutions
- Give a chance to get to know several solutions to choose the most appropriate

- The best lesson is to practice
  - test four tools on the same project
  - the project will be large enough to show features of given solution

Short high-end description of the project

- Image for RPi0 with X server, allowing to run chromium browser



- Changes in `local.conf` to set machine, distro and some variables

- Used layers
  - meta-openembedded and poky - basic layers used in builds, each of them contains smaller layers and we will use some of them (e.g. meta-networking)
  - meta-swupdate - provide possibility to create files to update system
  - meta-webkit - some useful tools including benchmark test for browser
  - meta-raspberrypi - RPi layer, provides, among others, machine config for the used hardware
  - meta-presentation - custom layer that allows you to make changes to existing recipes (e.g. dnsmasq) to be applied to the target image
- Patch one of the used layers

## Presentation goal summary

- Compare various of tools on the same project
- Listed project requirements cover most of the popular tasks that developer need to face when preparing layers
- Order of tested tools is random, each of them will end with a pros and cons of given solution
- Every custom configuration file or script needs to be placed in our custom layer which is meta-presentation

- git submodule - one of the git's internal tools
- Description from <https://git-scm.com/>

It often happens that while working on one project, you need to use another project from within it. Perhaps it's a library that a third party developed or that you're developing separately and using in multiple parent projects. A common issue arises in these scenarios: you want to be able to treat the two projects as separate yet still be able to use one from within the other.

- Doesn't this description match perfectly the layers used in Yocto?
  - layers should be separated
  - it should be easy to use more than one layer at a time

- We should add other layers as submodules to our custom layer
- Adding submodules (meta layers) is done via the command `git submodule <URL>`, to add poky as submodule type

```
$ git submodule add https://git.yoctoproject.org/git/poky
```

- After adding all of them, commit changes

```
$ git commit -s -m "add needed meta-layers as submodules"
[git_submodules 0615242e8318] add needed meta-layers as submodules
6 files changed, 20 insertions (+)
create mode 100644 .gitmodules
create mode 160000 meta-openembedded
create mode 160000 meta-raspberrypi
create mode 160000 meta-swupdate
create mode 160000 meta-webkit
create mode 160000 poky
```

- Note special 160000 mode for files

- By default, added submodules will be checked out on master
- To set correct refspec, cd to any one of them, make git checkout and commit changes

```
$ cd poky
$ git checkout 88c6be81a5fbed098999fbef5576c5e0bb90cc21
$ cd ..
$ git add .
$ git commit -s -m "set correct refspec"
[git_submodules ca0f9be2a11d] set correct refspec
1 file changed, 1 insertions(+), 1 deletions(-)
$ git show
commit ca0f9be2a11d5d6de351dfef6b69884617041542 (HEAD -> git_submodules)
Author: Tomasz Żyjewski <tomasz.zyjewski@3mdeb.com>
Date:   Mon Oct 26 09:36:21 2020 +0100
```

set correct refspec

Signed-off-by: Tomasz Żyjewski <tomasz.zyjewski@3mdeb.com>

```
diff --git a/poky b/poky
index 4e4a302e37ac..88c6be81a5fb 160000
--- a/poky
+++ b/poky
@@ -1,1 @@
-Subproject commit 4e4a302e37ac06543e9983773cdeb4caf7728330d
+Subproject commit 88c6be81a5fbed098999fbef5576c5e0bb90cc21
```

- Apply custom patch on one of the used layers (poky)
- Needs to be done manually, git submodules do not support any kind of patching layers from local files

```
$ cp patches/0001-add-removed-classes.patch poky/
$ cd poky/
$ git apply 0001-add-removed-classes.patch
$ git st
HEAD detached at 88c6be81a5fb
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   meta/classes/image_types_wic.bbclass
        modified:   scripts/lib/wic/plugins/imager/direct.py

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        meta/classes/bluetooth.bbclass
        meta/classes/gnome.bbclass
```

- Modification of `bblayers.conf` and `local.conf` files
  - `git submodule` does not allow to make those changes automatically
- To use custom configuration files we need to
  - run `oe-init-build-env` first to create build environment and copy our custom configuration files **OR**
  - run `oe-init-build-env` with special `TEMPLATECONF` variable pointed to our custom configuration files
- Either way, this needs to be done every time someone starts working with project, possibility to make errors

## Summary of git submodules

### Pros

- Does not require additional dependencies, most developers use git, right?
- Easy to use

### Cons

- Limited to downloading selected layers and revision settings
- Preparation of the environment requires manual work or some trickery related to writing scripts

- repo - tool built on top of Git, comes from AOSP
- <https://gerrit.googlesource.com/git-repo/>
- Helps to manage many repositories, set correct revisions
- repo command is a Python script, the easiest way to use it is to download and add it to PATH variable

```
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           %             0         0      150k         0 --:--:-- --:--:-- --:--:--  151k
$ chmod a+x ~/bin/repo
$ repo --help
usage: repo COMMAND [ARGS]
```

repo is not yet installed. Use "repo init" to install it here.

The most commonly used repo commands are:

init	Install repo in the current working directory
help	Display detailed help on a command

For access to the full online help, install repo ("repo init").

- repo works with manifest files
- Our custom layer should contain default.xml file with list of needed meta layers. It could look as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest>
  <default remote="poky" revision="default"/>

  <remote name="poky" fetch="git://git.yoctoproject.org" />
  <remote name="rpi" fetch="git://git.yoctoproject.org" />
  <remote name="pres" fetch="ssh://github.com/Tomasz" />

  <project name="poky"
    revision="88c6be81a5fbed098999fbef5576c5e0bb90cc21"
    remote="poky"/>
  <project name="meta-raspberrypi"
    revision="9d0935c9bb1309431e62b8f8341eb503653e5ff5"
    remote="rpi"/>
  <project name="meta-presentation"
    revision="6cb191164259ee179163bfccb13a1dbac3c136ba"
    remote="pres" />
</manifest>
```

- Ssh may be needed when fetching from private repositories

- To initialize new project, create new directory and call `repo init`

```
$ mkdir yocto_project && cd yocto_project  
$ repo init -u <URL> -b <branch>
```

- After that run `repo sync`, all layers will be cloned and checked out to the correct revision
- Custom patch again need to be applied manually
- The same applies to modifying the configuration files, the only option remains is to copy them from the custom layer or a special call to `oe-init-build-env`
- Can cause some errors and need to be done every time

## Summary of repo

### Pros

- Has many built-in options that allow you to control what you want to clone, set the revision etc.
- Based on a manifest file downloaded from anywhere (it can be a custom meta-layer), which briefly and accurately describes the elements (layers) included in the project
- Popular solution, often used, which means that it is also well tested

### Cons

- Requires some preparation before use, for example installing a script to PATH
- As with `git submodules`, modifying anything other than the list of layers in the appropriate revision is impossible, requires writing additional scripts

- combo-layer - command-line utility
- <https://wiki.yoctoproject.org/wiki/Combo-layer>
- Creates one mega combo layer that is combination of several layers
- Script available from poky

```
$ ./combo-layer -h
Usage: combo-layer [options] action
Create and update a combination layer repository from multiple component repositories.
Action:
  init                initialise the combo layer repo
  update [components] get patches from component repos and apply them to the combo repo
  pull [components]   just pull component repos only
  splitpatch [commit] generate commit patch and split per component, default commit is HEAD
Options:
  --version            show program's version number and exit
  -h, --help          show this help message and exit
  -c CONFFILE, --conf=CONFFILE
                      specify the config file (conf/combo-layer.conf is the
                      default).
  -i, --interactive   interactive mode, user can edit the patch list and
                      patches
  -D, --debug          output debug information
  -n, --no-pull        skip pulling component repos during update
  --hard-reset        instead of pull do fetch and hard-reset in component
                      repos
  -H, --history        import full history of components during init
```

- Add combo-layer script and combo-layer.conf file to custom layer
- Entry for each layer can look as follows

```
[meta-webkit]
src_uri = https://github.com/Igalia/meta-webkit.git
local_repo_dir = /home/tzyjewski/yocto_project/sources/meta-webkit
dest_dir = meta-webkit
branch = master
last_revision = 7bba8b0f806b10912f5d427d19867d017fc3aa34
file_exclude = recipes-browser recipes-devtools recipes-extended recipes-flatpak recipes-graphics
```

- mandatory variables
  - src\_uri
  - local\_repo\_dir
  - last\_revision
- optional variables
  - branch
  - file\_exclude
  - more...

- Prepare mega layer, copy repository with configuration file and run `init`

```
$ ./meta-presentation/scripts/combo-layer init -c meta-presentation/conf/combo-layer.conf
```

- `combo-layer` not only clones given layers, it adds them to one repository, now we can commit all of them in one repo
- No options to automatically apply local patches to cloned layers
- `combo-layer` similar to the previous tools, limited to setting layers, unable to customize `local.conf` or `bblayers.conf`

## Summary of combo-layer

### Pros

- Transparent configuration file for selecting layers and their contents
- Powerful in case of customizing used layers
- Self-contained repository

### Cons

- Self-contained repository
- Reviews might be time consuming, if there are many changes in many layers

- kas - provides an easy mechanism to setup bitbake based projects
- <https://github.com/siemens/kas>
- Set layers, create settings, prepare environment, initiate build process
- Download kas-docker script to PATH (need docker installed)

```
$ wget -O ~/bin/kas-docker https://raw.githubusercontent.com/siemens/kas/1.0/kas-docker
$ chmod +x ~/bin/kas-docker
```

- kas-docker script

```
$ kas-docker
Usage: /home/tzyjewski/bin/kas-docker [OPTIONS] { build | shell } [KASOPTIONS] KASFILE
       /home/tzyjewski/bin/kas-docker [OPTIONS] clean
Positional arguments:
build                  Check out repositories and build target.
shell                 Run a shell in the build environment.
clean                 Clean build artifacts, keep downloads.
Optional arguments:
--isar                Use kas-isar container to build Isar image.
--docker-args         Additional arguments to pass to docker forrunning the build.
-v                   Print operations.
--ssh-dir             Directory containing SSH configurations,
                       avoid /home/tzyjewski/.ssh unless you fully trust the container.
--no-proxy-from-env   Do not inherit proxy settings from environment.
```

- kas-docker runs with kas configuration files, they describe used meta layers and changes to the configuration files
- Entry for a given layer

```
meta-raspberrypi:  
  url: https://git.yoctoproject.org/git/meta-raspberrypi  
  refspec: 9d0935c9bb1309431e62b8f8341eb503653e5ff5
```

- kas configuration files can be stored in custom layer, to start new project we usually create new directory, clone custom layer and start kas-docker with command

```
kas-docker build meta-presentation/kas.yml
```

- All layers will be cloned and checked out inside directory from which we run kas-docker

- kas configuration files allows to create default build settings
  - It is possible via `bblayers_conf_header:` and `local_conf_header:` sections, first will modify top of `bblayers.conf` file and the second `local.conf` file
  - BBLAYERS will be generated based of the layer entries from kas file
  - Distro and machine can be set via `distro` and `machine` variables in kas file
- 
- generated `bblayers.conf`
  - generated `local.conf`

```
$ cat build/conf/bblayers.conf
# standard
POKY_BBLAYERS_CONF_VERSION = "2"
BBPATH = "${TOPDIR}"
BBFILES ?= ""
LICENSE_FLAGS_WHITELIST = "commercial"
BBLAYERS ?= " \
/repo/ \
/work/meta-openembedded/meta-multimedia \
/work/meta-openembedded/meta-networking \
...
```

```
$ cat build/conf/local.conf
# debug-tweaks
EXTRA_IMAGE_FEATURES = "debug-tweaks"
...
# standard
CONF_VERSION = "1"
PACKAGE_CLASSES = "package_rpm"
SDKMACHINE = "x86_64"
USER_CLASSES = " \
buildstats \
image-mklibs \
image-prelink"
PATCHRESOLVE = "noop"
MACHINE ??= "raspberrypi0-wifi"
DISTRO ??= "pres-x11"
```

- kas configuration file also allows to patch cloned meta layers
- Responsible fragment of the configuration file

```
poky:
  url: https://git.yoctoproject.org/git/poky
  refspec: 88c6be81a5fbbed098999fbef5576c5e0bb90cc21
  layers:
    meta:
    meta-poky:
    meta-yocto-bsp:
  patches:
    removed-classes:
      repo: meta-presentation
      path: patches/0001-add-removed-classes.patch
```

## Summary of kas

### Pros

- Allows a large customization of the build environment
- Supports bundling the build configuration with a layer
- Actually the only tool that properly prepares `local.conf` and `bblayers.conf` files
- Prepares build environments

### Cons

- When using the `kas-docker` script, installation of the docker is required

- Every tool was presented with pros and cons
- Which one to use? you decide!
- ... but first, check all of them!
- Personally?
  - kas
  - repo
  - git submodules
  - combo-layer

## Q&A